scan manual page - Tcl Built-In Commands

tcl.tk/man/tcl/TclCmd/scan.htm

NAME

scan — Parse string using conversion specifiers in the style of sscanf

SYNOPSIS

scan *string format* ?*varName varName ...*?

INTRODUCTION

This command parses substrings from an input string in a fashion similar to the ANSI C **sscanf** procedure and returns a count of the number of conversions performed, or -1 if the end of the input string is reached before any conversions have been performed. *String* gives the input to be parsed and *format* indicates how to parse it, using **%** conversion specifiers as in **sscanf**. Each *varName* gives the name of a variable; when a substring is scanned from *string* that matches a conversion specifier, the substring is assigned to the corresponding variable. If no *varName* variables are specified, then **scan** works in an inline manner, returning the data that would otherwise be stored in the variables as a list. In the inline case, an empty string is returned when the end of the input string is reached before any conversions have been performed.

DETAILS ON SCANNING

Scan operates by scanning *string* and *format* together. If the next character in *format* is a blank or tab then it matches any number of white space characters in *string* (including zero). Otherwise, if it is not a **%** character then it must match the next character of *string*. When a **%** is encountered in *format*, it indicates the start of a conversion specifier. A conversion specifier contains up to four fields after the **%**: a XPG3 position specifier (or a ***** to indicate the converted value is to be discarded instead of assigned to any variable); a number indicating a maximum substring width; a size modifier; and a conversion character. All of these fields are optional except for the conversion character. The fields that are present must appear in the order given above.
When **scan** finds a conversion specifier in *format*, it first skips any white-space characters in *string* (unless the conversion character is **[** or **c**). Then it converts the next input characters according to the conversion specifier and stores the result in the variable given by the next argument to **scan**.

OPTIONAL POSITIONAL SPECIFIER

If the **%** is followed by a decimal number and a **$**, as in "**%2$d**", then the variable to use is not taken from the next sequential argument. Instead, it is taken from the argument indicated by the number, where 1 corresponds to the first *varName*. If there are any positional specifiers in *format* then all of the specifiers must be positional. Every *varName*

on the argument list must correspond to exactly one conversion specifier or an error is generated, or in the inline case, any position can be specified at most once and the empty positions will be filled in with empty strings.

OPTIONAL SIZE MODIFIER

The size modifier field is used only when scanning a substring into one of Tcl's integer values. The size modifier field dictates the integer range acceptable to be stored in a variable, or, for the inline case, in a position in the result list. The syntactically valid values for the size modifier are **h**, **L**, **l**, and **ll**. The **h** size modifier value is equivalent to the absence of a size modifier in the the conversion specifier. Either one indicates the integer range to be stored is limited to the same range produced by the **int()** function of the **expr** command. The **L** size modifier is equivalent to the **l** size modifier. Either one indicates the integer range to be stored is limited to the same range produced by the **wide()** function of the **expr** command. The **ll** size modifier indicates that the integer range to be stored is unlimited.

MANDATORY CONVERSION CHARACTER

The following conversion characters are supported:

d

The input substring must be a decimal integer. It is read in and the integer value is stored in the variable, truncated as required by the size modifier value.

o

The input substring must be an octal integer. It is read in and the integer value is stored in the variable, truncated as required by the size modifier value.

x or X

The input substring must be a hexadecimal integer. It is read in and the integer value is stored in the variable, truncated as required by the size modifier value.

b

The input substring must be a binary integer. It is read in and the integer value is stored in the variable, truncated as required by the size modifier value.

u

The input substring must be a decimal integer. The integer value is truncated as required by the size modifier value, and the corresponding unsigned value for that truncated range is computed and stored in the variable as a decimal string. The conversion makes no sense without reference to a truncation range, so the size modifier **ll** is not permitted in combination with conversion character **u**.

i

The input substring must be an integer. The base (i.e. decimal, octal, or hexadecimal) is determined by the C convention (leading 0 for octal; prefix 0x for hexadecimal). The integer value is stored in the variable, truncated as required by the size modifier value.

c

A single character is read in and its Unicode value is stored in the variable as an integer value. Initial white space is not skipped in this case, so the input substring may be a white-space character.

s
The input substring consists of all the characters up to the next white-space character; the characters are copied to the variable.

e or f or g or E or G
The input substring must be a floating-point number consisting of an optional sign, a string of decimal digits possibly containing a decimal point, and an optional exponent consisting of an **e** or **E** followed by an optional sign and a string of decimal digits. It is read in and stored in the variable as a floating-point value.

[*chars*]
The input substring consists of one or more characters in *chars*. The matching string is stored in the variable. If the first character between the brackets is a **]** then it is treated as part of *chars* rather than the closing bracket for the set. If *chars* contains a sequence of the form *a*-*b* then any character between *a* and *b* (inclusive) will match. If the first or last character between the brackets is a **-**, then it is treated as part of *chars* rather than indicating a range.

[^*chars*]
The input substring consists of one or more characters not in *chars*. The matching string is stored in the variable. If the character immediately following the **^** is a **]** then it is treated as part of the set rather than the closing bracket for the set. If *chars* contains a sequence of the form *a*-*b* then any character between *a* and *b* (inclusive) will be excluded from the set. If the first or last character between the brackets is a **-**, then it is treated as part of *chars* rather than indicating a range value.

n
No input is consumed from the input string. Instead, the total number of characters scanned from the input string so far is stored in the variable.

The number of characters read from the input for a conversion is the largest number that makes sense for that particular conversion (e.g. as many decimal digits as possible for **%d**, as many octal digits as possible for **%o**, and so on). The input substring for a given conversion terminates either when a white-space character is encountered or when the maximum substring width has been reached, whichever comes first. If a ***** is present in the conversion specifier then no variable is assigned and the next scan argument is not consumed.

DIFFERENCES FROM ANSI SSCANF

The behavior of the **scan** command is the same as the behavior of the ANSI C **sscanf** procedure except for the following differences:

1. **%p** conversion specifier is not supported.

2. For **%c** conversions a single character value is converted to a decimal string, which is then assigned to the corresponding *varName*; no substring width may be specified for this conversion.
3. The **h** modifier is always ignored and the **l** and **L** modifiers are ignored when converting real values (i.e. type **double** is used for the internal representation). The **ll** modifier has no **sscanf** counterpart.
4. If the end of the input string is reached before any conversions have been performed and no variables are given, an empty string is returned.

EXAMPLES

Convert a UNICODE character to its numeric value:

```
set char "x"
set value [scan $char %c]
```

Parse a simple color specification of the form *#RRGGBB* using hexadecimal conversions with substring sizes:

```
set string "#08D03F"
scan $string "#%2x%2x%2x" r g b
```

Parse a *HH:MM* time string, noting that this avoids problems with octal numbers by forcing interpretation as decimals (if we did not care, we would use the **%i** conversion instead):

```
set string "08:08"   ;# *Not* octal!
if {[scan $string "%d:%d" hours minutes] != 2} {
    error "not a valid time string"
}
# We have to understand numeric ranges ourselves...
if {$minutes < 0 || $minutes > 59} {
    error "invalid number of minutes"
}
```

Break a string up into sequences of non-whitespace characters (note the use of the **%n** conversion so that we get skipping over leading whitespace correct):

```
set string " a string {with braced words} + leading space "
set words {}
while {[scan $string %s%n word length] == 2} {
    lappend words $word
    set string [string range $string $length end]
}
```

Parse a simple coordinate string, checking that it is complete by looking for the terminating character explicitly:

```
set string "(5.2,-4e-2)"
# Note that the spaces before the literal parts of
# the scan pattern are significant, and that ")" is
# the Unicode character \u0029
if {
    [scan $string " (%f ,%f %c" x y last] != 3
    || $last != 0x0029
} then {
    error "invalid coordinate string"
}
puts "X=$x, Y=$y"
```

An interactive session demonstrating the truncation of integer values determined by size modifiers:

```
% set tcl_platform(wordSize)
4
% scan 20000000000000000000 %d
2147483647
% scan 20000000000000000000 %ld
9223372036854775807
% scan 20000000000000000000 %lld
20000000000000000000
```